TRADUCTION FRANÇAISE DU SUJET WEBSERV DE 42

Chapter I

Introduction

Le Hypertext Transfer Protocol (HTTP) est un protocole applicatif destiné aux systèmes d'information distribués, collaboratifs et hypermédias.

HTTP constitue la base de la communication de données sur le World Wide Web, où les documents hypertextes contiennent des hyperliens vers d'autres ressources accessibles facilement, par exemple via un clic de souris ou un écran tactile dans un navigateur web.

HTTP a été développé pour prendre en charge les fonctionnalités de l'hypertexte et soutenir la croissance du World Wide Web.

La fonction principale d'un serveur web est de stocker, traiter et délivrer des pages web aux clients. La communication client-serveur s'effectue via le protocole HTTP. Les pages transmises sont généralement des documents HTML pouvant inclure des images, feuilles de style et scripts.

Un site à fort trafic peut utiliser plusieurs serveurs web répartissant la charge entre plusieurs machines.

Un user agent (navigateur ou robot) initie la communication en demandant une ressource via HTTP. Le serveur répond avec le contenu ou une erreur. La ressource est souvent un fichier réel ou le résultat d'un programme, mais pas toujours.

HTTP permet aussi aux clients d'envoyer des données, notamment via les formulaires et l'upload de fichiers.

Chapter II

General rules

• Votre programme ne doit jamais planter, même en cas de manque de mémoire. Sinon : note 0.

• Vous devez fournir un Makefile qui compile vos sources sans relinking inutile.

• Règles obligatoires : $(NAME), all, clean, fclean, re.

• Compilation : c++ -Wall -Wextra -Werror.

• Code conforme au standard C++98 (-std=c++98).

• Utilisation encouragée des fonctionnalités C++ (ex. plutôt que).

• Aucune bibliothèque externe ni Boost autorisée.

Chapter IV

Mandatory part

Nom du programme : webserv

Fichiers à rendre : Makefile, *.{h,hpp}, *.cpp, *.tpp, *.ipp, fichiers de configuration.

Fonctions externes autorisées : execve, pipe, strerror, gai_strerror, errno, dup, dup2, fork, socketpair, htons, htonl, ntohs, ntohl, select, poll, epoll, kqueue, socket, accept, listen, send, recv, chdir, bind, connect, getaddrinfo, freeaddrinfo, setsockopt, getsockname, getprotobyname, fcntl, close, read, write, waitpid, kill, signal, access, stat, open, opendir, readdir, closedir.

Description : Un serveur HTTP en C++98.

Lancement :

./webserv [configuration file]

poll() est mentionné mais vous pouvez utiliser select(), kqueue() ou epoll().

Lire les RFC HTTP et tester avec telnet / NGINX est recommandé.

HTTP/1.0 est une référence conseillée mais non imposée.

IV.1 Requirements

• Fichier de configuration obligatoire, fourni en argument ou via un chemin par défaut.

• Impossible d’execve un autre serveur web.

• Serveur toujours non bloquant, gérant correctement les déconnexions.

• Un seul poll() (ou équivalent) doit gérer toutes les I/O (lecture, écriture, listen).

• Interdiction d’effectuer read/write sans poll() préalable.

• Interdiction d’utiliser errno après read/write pour ajuster le comportement.

• poll() non requis pour les fichiers classiques (read/write possibles directement).

• Une requête ne doit jamais rester bloquée indéfiniment.

• Compatibilité avec les navigateurs web standard.

• NGINX peut servir de référence pour comparer les headers et comportements.

• Codes HTTP exacts obligatoires.

• Pages d’erreur par défaut si aucune n’est fournie.

• fork uniquement pour les CGI.

• Le serveur doit pouvoir : servir un site statique, gérer l'upload, supporter GET / POST / DELETE.

• Test de charge obligatoire.

• Support de plusieurs ports.

IV.2 macOS only

• macOS gère write() différemment : fcntl() autorisé uniquement avec F_SETFL, O_NONBLOCK, FD_CLOEXEC.

• Descripteurs en mode non bloquant obligatoires.

IV.3 Configuration file

Inspiré de la section 'server' de NGINX.

Le fichier de configuration doit permettre :

• Définir toutes les interfaces:ports.

• Définir les pages d'erreur.

• Fixer la taille maximale des bodies.

• Définir des règles par route : méthodes autorisées, redirections, dossier racine, listing autorisé ou non, fichier par défaut, upload autorisé + répertoire de stockage.

• Gérer les CGI selon les extensions (.php, etc.)

Notes CGI :

• Le CGI doit recevoir la requête complète via les variables d'environnement.

• Pour les requêtes chunked, le serveur doit dé-chunker avant d'envoyer au CGI.

• Pour la sortie CGI : si pas de content_length $\rightarrow$ EOF marque la fin.

• Le CGI doit être exécuté dans le bon répertoire.

• Votre serveur doit supporter au moins un CGI.

Vous devez fournir des fichiers de configuration pour démontrer toutes les fonctionnalités.

Un testeur est fourni (facultatif).

La résilience est essentielle : votre serveur doit rester opérationnel.

Écrivez vos propres tests en Python, Go, C ou C++.